Title : SamACO: Variable Sampling Ant Colony Optimization Algorithm for Continuous Optimization

Authors : Xiao-Min Hu[*], *Student Member, IEEE*

Jun Zhang[*], *Senior Member, IEEE* (corresponding author)

Henry S.-H. Chung[+], *Senior Member, IEEE*

Yun Li[※], *Member, IEEE*

Ou Liu[o]

Affiliation: [*] Sun Yat-Sen University, Key Laboratory of Digital Life（Sun Yat-sen University）, Ministry of Education

[+] City University of Hong Kong

[※]University of Glasgow

[o]Hong Kong Polytechnic University, Kowloon, Hong Kong.

Address : Department of Computer Science,

Sun Yat-Sen University,

China

Email : junzhang@ieee.org

SamACO: Variable Sampling Ant Colony Optimization Algorithm for Continuous Optimization

Xiao-Min Hu, *Student Member, IEEE*, Jun Zhang, *Senior Member, IEEE*, Henry S.-H. Chung, *Senior Member, IEEE*, Yun Li, *Member, IEEE*, and Ou Liu

Abstract –An ant colony optimization (ACO) algorithm offers algorithmic techniques for optimization by simulating the foraging behavior of a group of ants to perform incremental solution constructions and to realize a pheromone laying-and-following mechanism. Although ACO is first designed for solving discrete (combinatorial) optimization problems, the ACO procedure is also applicable to continuous optimization. This paper presents a new way of extending ACO to solving continuous optimization problems, by focusing on continuous variable sampling as key to transforming ACO from discrete optimization to continuous optimization. The proposed SamACO algorithm consists of three major steps, i.e., the generation of candidate variable values for selection, the ants' solution construction, and the pheromone update process. The distinct characteristics of SamACO are the cooperation of a novel sampling method for discretizing the continuous search space and an efficient incremental solution construction method based on the sampled values. The performance of SamACO is tested using continuous numerical functions with unimodal and multimodal features. Compared with some state-of-the-art algorithms, including traditional ant-based algorithms and representative computational intelligence algorithms for continuous optimization, the performance of SamACO is seen competitive and promising.

Index terms –Ant algorithm, ant colony optimization (ACO), ant colony system (ACS), continuous optimization, function optimization, local search, numerical optimization

I. Introduction

Simulating the foraging behavior of ants in nature, ant colony optimization (ACO) algorithms [1][2] are a class of swarm intelligence algorithms originally developed to solve discrete (combinatorial) optimization problems, such as traveling salesman problems (TSPs) [3][4], multiple knapsack problems (MKPs) [5], network routing problems [6][7], scheduling problems [8]-[10], and circuit design problems [11]. When solving these problems, pheromones are deposited by ants on nodes or links connecting the nodes in a construction graph [2]. Here, the ants in the algorithm represent stochastic constructive procedures for building solutions. The pheromone, used as a metaphor for an odorous chemical substance that real ants deposit and smell while walking, has similar effects on biasing the ants' selection of nodes in the algorithm. Each node represents a candidate component value, which belongs to a finite set of discrete decision variables. Based on the pheromone values, the ants in the algorithm probabilistically select the component values to construct solutions.

For continuous optimization, however, decision variables are defined in the continuous domain and hence the number of possible candidate values would be infinite for ACO. Therefore, how to utilize pheromones in the continuous domain for guiding ants' solution construction is an important problem to solve in extending ACO to continuous optimization. According to the uses of the pheromones, there are three types of ant-based algorithms for solving continuous optimization problems in the literature.

The first type does not use pheromones but use other forms of implicit or explicit cooperation. For example, API [12] simulates the foraging behavior of *Pachycondyla apicalis* ants, which use visual landmarks but not pheromones to memorize the positions and search the neighborhood of the hunting sites.

The second type of ant-based algorithms places pheromones on the points in the search space. Each point is in effect a complete solution, indicating a region for the ants to perform

local neighborhood search. This type of ant-based algorithms generally hybridizes with other algorithms for maintaining diversity. The continuous ACO (CACO) [13]-[15] is a combination of the ants' pheromone mechanism and a genetic algorithm. The continuous interacting ant colony (CIAC) algorithm proposed by Dréo and Siarry [16] uses both pheromone information and the ants' direct communications to accelerate the diffusion of information. The continuous orthogonal ant colony (COAC) algorithm proposed by Hu *et al.* [17] adopts an orthogonal design method and a global pheromone modulation strategy to enhance the search accuracy and efficiency. Other methods such as hybridizing a Nelder-Mead simplex algorithm [18] and using a discrete encoding [19] have also been proposed. Since pheromones are associated with the entire solutions instead of components in this type of algorithms, no incremental solution construction is performed during the optimization process.

The third type of algorithms follows the ACO framework, i.e., the ants in the algorithms construct solutions incrementally biased by pheromones on components. Socha [20] extended the traditional ACO for solving both continuous and mixed discrete-continuous optimization problems. Socha and Dorigo [21] later improved their algorithm and referred the resultant algorithm to ACO_R, where an archive was used to preserve the k best solutions found thus far. Each solution variable value in the archive is considered as the center of a Gaussian probability density function (PDF). Pheromones in ACO_R are implicit in the generation of Gaussian PDFs. When constructing a solution, a new variable value is generated according to the Gaussian distribution with a selected center and a computed variance. The fundamental idea of ACO_R is the shift from using a discrete probability distribution to using a continuous PDF. The sampling behavior of ACO_R is a kind of probabilistic sampling, which samples a PDF [21]. Similar realizations of this type are also reported in [22]-[28].

Different from sampling a PDF, the SamACO algorithm proposed in this paper is based

on the idea of sampling candidate values for each variable and selecting the values to form solutions. The motivation for this work is that a solution of a continuous optimization problem is in effect a combination of feasible variable values, which can be regarded as a solution ‘path’ walked by an ant. The traditional ACO is good at selecting promising candidate component values to form high-quality solutions. Without loss of the advantage of the traditional ACO, a means to sample promising variable values from the continuous domain and to use pheromones on the candidate variable values to guide the ants’ solution construction is developed in SamACO.

A distinctive characteristic of SamACO is a novel sampling method for discretizing the continuous search space and an efficient method for incremental solution construction based on the sampled variable values. In SamACO, the sampling method possesses the feature of balancing memory, exploration, and exploitation. By preserving variable values from the best solutions constructed by the previous ants, promising variable values are inherited from the last iteration. Diversity of the variable values is maintained by exploring a small number of random variable values. High solution accuracy is achieved by exploiting new variable values surrounding the best-so-far solution by a dynamic exploitation process. If a high-quality solution is constructed by the ants, the corresponding variable values will receive additional pheromones, so that the latter ants can be attracted to select the values again.

Differences between the framework of ACO in solving discrete optimization problems and the proposed SamACO in solving continuous optimization problems will be detailed in the next section. The performance of SamACO in solving continuous optimization problems will be validated by testing benchmark numerical functions with unimodal and multimodal features. The results are not only compared with the aforementioned ant-based algorithms, but also with some representative computational intelligence algorithms, e.g., CLPSO [29], FEP [30], and CMA-ES [31].

The rest of the paper is organized as follows. Section II firstly presents the traditional ACO framework for discrete optimization. The SamACO framework is then introduced in a more general way. Section III describes the implementation of the proposed SamACO algorithm for continuous optimization. Parameter analysis of the proposed algorithm is made in Section IV. Numerical experimental results are presented in Section V for analyzing the performance of the proposed algorithm. Conclusions are drawn in Section VI.

II. ACO Framework for Discrete and Continuous Optimization

A. Traditional ACO Framework for Discrete Optimization

Before introducing the traditional ACO framework, a discrete minimization problem is firstly defined as in [2][21].

Definition 1: A discrete minimization problem Π is denoted as (S, f, Ω), where

- S is the search space defined over a finite set of discrete decision variables (solution components) X_i with values $x_i^{(j)} \in \mathbf{D}_i = \{x_i^{(1)}, x_i^{(2)}, ..., x_i^{(|\mathbf{D}_i|)}\}$, $i = 1,2,...,n$, n being the number of decision variables.
- $f : S \to \Re$ is the objective function. Each candidate solution $\boldsymbol{x} \in S$ has an objective function value $f(\boldsymbol{x})$. The minimization problem is to search for a solution $\boldsymbol{x}^* \in S$ that satisfies $f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$, $\forall \boldsymbol{x} \in S$.
- Ω is the set of constraints that the solutions in S must satisfy.

The most significant characteristic of ACO is the use of pheromones and the incremental solution construction [2]. The basic framework of the traditional ACO for discrete optimization is shown in Fig. 1. Besides the initialization step, the traditional ACO comprises the ant's solution construction, an optional local search, and the pheromone update. The three processes iterate until the termination condition is satisfied.

When solving discrete optimization problems, solution component values or the links

between the values are associated with pheromones. If component values are associated with pheromones, a component-pheromone matrix **M** can be generated. The pheromone value $\tau_i^{(j)}$ reflects the desirability for adding the component value $x_i^{(j)}$ to the solution, $i = 1,2,...,n$, $j = 1,2,...,|\mathbf{D}_i|$.

$$\mathbf{M} = \begin{bmatrix} \{x_1^{(1)}, \tau_1^{(1)}\} & \{x_2^{(1)}, \tau_2^{(1)}\} & \cdots & \{x_n^{(1)}, \tau_n^{(1)}\} \\ \{x_1^{(2)}, \tau_1^{(2)}\} & \{x_2^{(2)}, \tau_2^{(2)}\} & \cdots & \{x_n^{(2)}, \tau_n^{(2)}\} \\ \vdots & \vdots & \ddots & \vdots \\ \{x_1^{(|\mathbf{D}_1|)}, \tau_1^{(|\mathbf{D}_1|)}\} & \{x_2^{(|\mathbf{D}_2|)}, \tau_2^{(|\mathbf{D}_2|)}\} & \cdots & \{x_n^{(|\mathbf{D}_n|)}, \tau_n^{(|\mathbf{D}_n|)}\} \end{bmatrix} \quad (1)$$

If the links between the values are associated with pheromones, each $< x_i^{(j)}, x_u^{(l)} >$ tuple will be assigned a pheromone value $\tau_{i,u}^{(j,l)}$, $j = 1,2,...,|\mathbf{D}_i|$, $l = 1,2,...,|\mathbf{D}_u|$, $i,u = 1,2,...,n$, $i \neq u$. Since the treatment of placing pheromones on links or components is similar, the rest of this paper will focus on the case where pheromones are on components.

As the number of component values is finite in discrete optimization problems, pheromone update can be applied directly to the $\tau_i^{(j)}$ in (1) for increasing or reducing the attractions of the corresponding component values to the ants. The realization of the pheromone update process is the main distinction among different ACO variants in the literature, such as ant system (AS) [3], rank-based ant system (AS_{rank}) [32], Max-Min ant system (MMAS) [33], and ant colony system (ACS) [4].

B. Extended ACO Framework for Continuous Optimization

Different from discrete optimization problems, a continuous optimization problem is defined as follows [2][21].

Definition 2: A continuous minimization problem Π is denoted as (S, f, Ω), where

- S is the search space defined over a finite set of continuous decision variables (solution components) X_i, $i = 1,2,...,n$, with values $x_i \in [l_i, u_i]$, l_i and u_i representing the lower

and upper bounds of the decision variable X_i respectively.

- The definitions of f and Ω are the same as in Definition 1.

The difference from the discrete optimization problem is that in continuous optimization for ACO the decision variables are defined in the continuous domain. Therefore, the traditional ACO framework needs to be duly modified.

In the literature, researchers such as Socha and Dorigo [21] proposed a method termed ACO_R to shift the discrete probability distribution in discrete optimization to a continuous probability distribution for solving the continuous optimization problem, using probabilistic sampling. When an ant in ACO_R constructs a solution, a Gram-Schmidt process is used for each variable to handle variable correlation. However, the calculation of the Gram-Schmidt process for each variable leads to a significantly higher computational demand. When the dimension of the objective function increases, the time used by ACO_R also increases rapidly. Moreover, although the correlation between different decision variables is handled, the algorithm may still converge to local optima, particularly when the values in the archive are very close to each other.

However, if a finite number of variable values are sampled from the continuous domain, the traditional ACO algorithms for discrete optimization can be used. This forms the basic idea of the SamACO framework proposed in this paper. The key for successful optimization now becomes how to sample promising variable values and use them to construct high-quality solutions.

III. The Proposed SamACO Algorithm for Continuous Optimization

This section presents the detailed implementation of the proposed SamACO algorithm for continuous optimization. The success of SamACO in solving continuous optimization problems depends on an effective variable sampling method and an efficient solution

construction process. The variable sampling method in SamACO maintains promising variable values for the ants to select, including variable values selected by the previous ants, diverse variable values for avoiding trapping, and variable values with high-accuracy. The ants' construction process selects promising variable values to form high-quality solutions by taking advantage of the traditional ACO method in selecting discrete components.

The SamACO framework for continuous optimization is shown in Fig. 2. Each decision variable X_i has k_i sampled values $x_i^{(1)}, x_i^{(2)}, ..., x_i^{(k_i)}$ from the continuous domain $[l_i, u_i]$, i=1,2,…,n. The sampled discrete variable values are then used for optimization by a traditional ACO process as in solving discrete optimization problems (DOPs). Fig. 3 illustrates the flowchart of the algorithm. The overall pseudocode of the proposed SamACO is shown in the Appendix I.

A. Generation of the Candidate Variable Values

The generation processes of the candidate variable values in the initialization step and in the optimization iterations are different. Initially, the candidate variable values are randomly sampled in the feasible domain as

$$x_i^{(j)} = l_i + \frac{u_i - l_i}{m + \vartheta}(j - 1 + rand_i^{(j)}) \tag{2}$$

where $(m + \vartheta)$ is the initial number of candidate values for each variable i, $rand_i^{(j)}$ is a uniform random number in [0,1], $i = 1,2,...,n$, $j = 1,2,...,m + \vartheta$.

During the optimization iterations, candidate variable values have four sources, i.e., the variable values selected by ants in the previous iteration, a dynamic exploitation, a random exploration, and a best-so-far solution. In each iteration, m ants construct m solutions, resulting in m candidate values for each variable for the next iteration. The best-so-far solution is then updated, representing the best solution that has ever been found. The dynamic

exploitation is applied to the best-so-far solution, resulting in g_i new candidate variable values near the corresponding variable values of the best-so-far solution for each variable X_i. Furthermore, a random exploration process generates Θ new values for each variable by discarding the worst Θ solutions that are constructed by the ants in the previous iterations. Suppose the worst Θ solutions are denoted by $\boldsymbol{x}^{(m-\Theta+1)}, \boldsymbol{x}^{(m-\Theta+2)},..., \boldsymbol{x}^{(m)}$. The new variable values for the solution $\boldsymbol{x}^{(j)}$ are randomly generated as

$$x_i^{(j)} = l_i + (u_i - l_i) \cdot rand_i^{(j)} \tag{3}$$

where $i = 1,2,...,n$, $j = m - \Theta + 1,...,m$. New values thus can be imported in the value group. To summarize, the composition of the candidate variable values for the ants to select is illustrated in Fig. 4. There are totally $(m + g_i + 1)$ candidate values for each variable X_i.

B. Dynamic Exploitation Process

The dynamic exploitation proposed in this paper is effective for introducing fine-tuned variable values into the variable value group. We use a radius r_i to confine the neighborhood exploitation of the variable value x_i, $i = 1,2,...,n$.

The dynamic exploitation is applied to the best-so-far solution $\boldsymbol{x}^{(0)} = (x_1^{(0)}, x_2^{(0)},..., x_n^{(0)})$, aiming at searching the vicinity of the variable value $x_i^{(0)}$ in the interval $[x_i^{(0)} - r_i, x_i^{(0)} + r_i]$, $i = 1,2,...,n$. The values of the variables in the best-so-far solution are randomly selected to be increased, unchanged, or reduced as

$$\hat{x}_i = \begin{cases} \min(x_i^{(0)} + r_i \cdot \sigma_i, u_i), & 0 \le q < 1/3 \\ x_i^{(0)}, & 1/3 \le q < 2/3 \\ \max(x_i^{(0)} - r_i \cdot \sigma_i, l_i), & 2/3 \le q < 1 \end{cases} \tag{4}$$

where $\sigma_i \in (0,1]$ and $q \in [0,1)$ are uniform random values, $i = 1,2,...,n$. Then the resulting solution $\hat{\boldsymbol{x}} = (\hat{x}_1, \hat{x}_2,..., \hat{x}_n)$ is evaluated. If the new solution is no worse than the recorded

best-so-far solution, we replace the best-so-far solution with the new solution. The above exploitation repeats for ϑ times. The new variable values that are generated by increasing or reducing a random step length are recorded as $x_i^{(j)}$, $j = m+1, m+2, ..., m+g_i$, $i = 1,2,...,n$, where g_i counts the number of new variable values in the dynamic exploitation process.

In each iteration, the radiuses adaptively change based on the exploitation result. If the best exploitation solution is no worse than the original best-so-far solution (Case 1), the radiuses will be extended. Otherwise (Case 2), the radiuses will be reduced.

$$r_i \leftarrow \begin{cases} r_i \cdot v_{\mathrm{e}}, & \text{Case 1} \\ r_i \cdot v_{\mathrm{r}}, & \text{Case 2} \end{cases} \tag{5}$$

where v_{e} $(v_{\mathrm{e}} \geq 1)$ is the radius extension rate, v_{r} $(0 < v_{\mathrm{r}} \leq 1)$ is the radius reduction rate. The initial radius value is set as $r_i = (u_i - l_i)/(2m)$, $i = 1,2,...,n$. The extension of the radiuses can import values in a distance further away from the original value, whereas the reduction of the radiuses can generate values with high accuracy near to the original value. The extension and reduction of radiuses aim at introducing values with different accuracy levels according to the optimization situation.

C. Ants' Solution Construction

After the candidate variable values are determined, *m* ants are dispatched to construct solutions. Each candidate variable value is associated with pheromones, which bias the ants' selection for solution construction. The index $l_i^{(k)}$ of the variable value selected by ant *k* for the *i*th variable is

$$l_i^{(k)} = \begin{cases} \arg\ \max\{\tau_i^{(1)}, \tau_i^{(2)}, ..., \tau_i^{(m)}\}, & \text{if } q < q_0 \\ L_i^{(k)}, & \text{otherwise} \end{cases} \tag{6}$$

where q is a uniform random value in [0,1), $i = 1,2,...,n$, $k = 1,2,...,m$. The parameter

$q_0 \in [0,1)$ controls whether an ant will choose the variable value with the highest pheromone from the *m* solutions generated in the previous iteration, or randomly choose an index $L_i^{(k)} \in \{0,1,...,m+g_i\}$ according to the probability distribution given in (7).

$$p_i^{(j)} = \frac{\tau_i^{(j)}}{\sum_{u=0}^{m+g_i} \tau_i^{(u)}}, \quad j = 0,1,...,m+g_i \tag{7}$$

The constructed solution of ant *k* is denoted $\boldsymbol{x}^{(k)} = (x_1^{(l_1^{(k)})}, x_2^{(l_2^{(k)})},..., x_n^{(l_n^{(k)})})$. An illustration of two ants constructing solutions is shown in Fig. 5. The variable values that are selected by each ant form a solution 'path'. Pheromones will then be adjusted according to the quality of these solution paths.

D. Pheromone Update

Initially, each candidate variable value is assigned an initial pheromone value T_0. After evaluating the *m* solutions constructed by ants, the solutions are sorted by their objective function values in an order from the best to the worst. Suppose the sorted solutions are arranged as $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)},..., \boldsymbol{x}^{(m)}$. The pheromones on the selected variable values are evaporated as

$$\tau_i^{(j)} \leftarrow (1-\rho) \cdot \tau_i^{(j)} + \rho \cdot T_{\min} \tag{8}$$

where $0 < \rho < 1$ is the pheromone evaporation rate, and $T_{\min}$ is the predefined minimum pheromone value, $i = 1,2,...,n$, $j = 1,2,...,m$.

The variable values in the best Ψ solutions have their pheromones reinforced as

$$\tau_i^{(j)} \leftarrow (1-\alpha) \cdot \tau_i^{(j)} + \alpha \cdot T_{\max} \tag{9}$$

where $0 < \alpha < 1$ is the pheromone reinforcement rate, and $T_{\max}$ is the predefined maximum pheromone value, $i = 1,2,...,n$, $j = 1,2,...,\Psi$, Ψ being the number of the high-quality

solutions which receive additional pheromones on the variable values. The pheromones on the variable values that are generated by the exploration process and the dynamic exploitation process are assigned as T_0. In each iteration, pheromone values on the variable values of the best-so-far solution are assigned equal to the pheromone values on the iteration best solution.

IV. Parameter Study of SamACO

It can be seen that the proposed SamACO involves several parameters. In this section, we will investigate the effects of these parameters on the proposed algorithm.

A. Effects of the Parameters in SamACO

1) Discarding Number Θ

The discarding number $\Theta \geq 1$ is used for maintaining diversity in the candidate variable values. In each iteration, the random exploration process replaces the Θ worst solutions constructed by ants. The more solutions are replaced, the higher diversity in the candidate variable values is. Diversity is important for avoiding stagnation, but a high discarding rate can slow down the convergence speed.

2) Elitist Number Ψ

Different from the discarding number Θ, the elitist number Ψ determines the best solutions constructed by ants to receive additional pheromones. The variable values with extra pheromone deposits have higher chances to be selected by ants. Therefore, the elitist number Ψ helps preserve promising variable values and accelerate the convergence speed.

Both of the parameters Θ and Ψ cannot be set too large. In fact, a small discarding number and a small elitist number are sufficient, because their effects accumulate in iterations.

3) Traditional Parameters in ACO

Similar to the ACS [4], parameters of SamACO to simulate the ants' construction

behavior include the number of ants *m*, parameter q_0, pheromone evaporation rate ρ, pheromone reinforcement rate α, initial pheromone value T_0, and lower and upper limits of pheromone values $T_{\min}$ and $T_{\max}$. In the literature, a great deal of work has been carried out on the parametric study for ACO, such as [4][34]-[37]. In the next subsection, we will use experiments to find suitable settings for these parameters.

4) Parameters in Dynamic Exploitation

The parameters in the dynamic exploitation includes the exploitation frequency ϑ, the radius reduction and extension rates v_{r} and v_{e}. The exploitation frequency ϑ controls the number of values to be sampled in the neighborhood of the best-so-far solution per iteration. Since each candidate variable value group is mainly composed of *m* variable values that are selected by the previous ants and the variable values from the local exploitation, the value of ϑ influences the selection probabilities of the *m* variable values. Furthermore, a large ϑ provides more candidate variable values surrounding the neighborhood of the best-so-far solution, but it may induce premature convergence. On the contrary, a small ϑ may not sample enough local variable values and thus the speed to approach the local optimum is slow. On the other hand, the radius reduction and extension rates v_{r} and v_{e} adapt the exploitation neighborhood range according to the solution quality. Because the radius extension is a reversal operation of the radius reduction, we set $v_{\mathrm{e}} = 1/v_{\mathrm{r}}$ in this paper.

B. Numerical Analysis on Parameter Settings

Some of the SamACO parameters have their default settings. They are set as $T_{\min} = T_0 = 0.1$, $T_{\max} = 1.0$, $\Theta = 1$, and $\Psi = 1$. There is generally no need to change these values. However, the performance of the proposed algorithm is more sensitive to other parameters.

According to our prior experiments on a set of unimodal and multimodal functions (listed in Appendix II) with n=30, the parameters $m = 20$, $\vartheta = 20$, $\rho = 0.5$, $\alpha = 0.3$, $q_0 = 0.1$, $v_r = 0.7$ are used for comparing the performance of SamACO with other algorithms. In the remainder of this section, we undertake a numerical experiment on f_6 (the shifted Schwefel's problem 1.2) for discussing the influence of these parameters. The algorithm terminates when the number of function evaluations (FEs) reaches 300000. Each parameter setting is tested for 25 times independently.

1) Correlations Between Ant Number m and Exploitation Frequency ϑ

The correlations between m=1,4,8,12,…,40 and ϑ=0,1,2,3,4,8,12,…,40 have been tested by fixing the values of the other parameters. Fig. 6 shows a contour graph representing the average number of FEs used for achieving an accuracy level smaller than $\varepsilon = f(\boldsymbol{x}) - f(\boldsymbol{x}^*) = 1$, where $f(\boldsymbol{x})$ and $f(\boldsymbol{x}^*)$ denote the solution value and the optimal solution value respectively. The smaller the number of FEs used, the better the performance of the proposed algorithm. It can be observed in Fig. 6 that SamACO is more sensitive in m than in ϑ. When the value of ϑ is fixed, a larger m consumes more FEs. Furthermore, when ϑ=0 and 1 (not shown in Fig. 6), SamACO cannot achieve any satisfactory results within the maximum number of FEs (300000) using any value of m. Therefore, sufficient amount of dynamic exploitation is crucial to achieving solutions with high accuracy.

2) Radius Reduction Rate v_r

The values of v_r=0,0.1,0.2,…,1 have been tested, with the other parameter values fixed. Fig. 7(a) illustrates a curve representing the average number of FEs used with different values of v_r on f_6. The curve shows a significant decreasing trend when v_r increases from 0 to 1. When v_r=0.7, the average number of FEs used is minimized. With a larger value of v_r, the reduction speed of radiuses is slower so that promising variable values is not so easy to be

omitted when exploiting the neighborhood. When v_r=0 ($v_e = 0$), the neighborhood radius is 0. When $v_r = v_e = 1$, the neighborhood radius is fixed as the initial value, which is too large to obtain solutions within the predefined accuracy level. In the above two cases SamACO cannot find satisfactory solutions.

3) Traditional ACO Parameters q_0, ρ, α

The values of 0, 0.1, 0.2,…, 1 have been assigned to q_0, ρ, α respectively for analyzing their influence to SamACO. The curves showing the average number of FEs used with different values of q_0, ρ, α are given in Figs. 7(b)-(d). It can be observed in Fig. 7(b) that an inclination for selecting the variable value having the largest pheromone from the *m* variable values (as in (6)) is beneficial, because SamACO uses smaller numbers of FEs when q_0=0.2 and 0.1 than that of q_0=0. However, the value of q_0 cannot be too large. Otherwise, the algorithm traps in local optima and needs a long time to jump out. From Figs. 7(c) and 7(d), the curves of ρ and α show similar trends. The algorithm has unsatisfactory performance when $\rho = 0$ or $\alpha = 0$, in which case local or global pheromone update does not take any effect. Therefore, pheromone updates are quite useful for the optimization process. To facilitate practical applications of SamACO to various problems, Table I lists the domains and default settings (denoted by DS) of the parameters (P) and concludes the sensitivity of SamACO based on the default settings.

V. Numerical Experiments

A. Experimental Setup

Sixteen benchmark numerical functions have been tested in this paper to validate the performance of the proposed SamACO. Appendix II lists these functions chosen from the literature [30][40]. Among them, functions f_1 to f_9 are unimodal, whereas f_{10} to f_{16} are

multimodal.

Three types of algorithms are used for comparing the performance of SamACO. The first type includes the ant-based algorithms such as CACO [15], COAC [17], and ACO_R [21]. The second type consists of the other well-known swarm intelligence algorithm as ACO, that is, particle swarm optimization (PSO) [29][38][39]. Different from ACO, PSO was originally developed for continuous optimization. An improved variant of PSO – the comprehensive learning particle swarm optimization (CLPSO) [29] is applied. The third type consists of representative algorithms that use an explicit probability-learning notion. In general, most computational intelligence algorithms have implicit or explicit probabilistic models that guide the optimization process. ACO uses pheromones as explicit guidance. Besides, algorithms such as evolutionary programming and evolution strategy also learn from explicit probabilistic models for generating new solutions. Here we use the fast evolutionary programming (FEP) [30] and the evolution strategy with covariance matrix adaptation (CMA-ES) [31] as representatives of the probability-learning algorithms.

Except for CMA-ES, algorithms in comparison with SamACO are programmed in C, according to the source code provided by the authors and the descriptions given in the references. The CMA-ES is programmed in MATLAB[1]. Parameters of the algorithms are set as the recommended values according to the references. Notice that all of the algorithms terminate when the number of FEs reaches 300000. Each function is tested for 25 times independently.

B. Results and Comparisons

1) Comparison With Other Ant-Based Algorithms

Table II tabulates the mean error values $f(\boldsymbol{x})-f(\boldsymbol{x}^*)$ and the corresponding standard

[1] The MATLAB code used, cmaes.m, Version 2.35, is available at http://www.bionik.tu-berlin.de/user/niko/formersoftwareversions.html

deviations (St. dev) on f_1 - f_{16} for SamACO, CACO, COAC, and ACO_R when n=30. A t-test is made based on the error values of 25 independent runs for showing whether SamACO is significantly better or worse than the other algorithms. If the same zero error values are obtained for the compared algorithms (i.e., f_1), the numbers of FEs that are required for achieving the zero error values are used by the t-test. A score 1, 0, or -1 is added when SamACO is significantly better than (b†), indifferent to (i), or significantly worse than (w‡) the compared algorithm for each function. In the table, the total scores corresponding to the other ant-based algorithms are all positive, meaning that the performance of SamACO is generally better than those algorithms.

Table III shows the average number of FEs that are required to achieve accuracy levels smaller than $\varepsilon_1 = 1$ and $\varepsilon_2 = 10^{-6}$. The symbol × denotes that the results cannot reach the predefined accuracy level ε within maximum 300000 FEs, whereas '%ok' stands for the successful percentage over 25 independent runs. Only are successful runs used for calculating the average number of FEs and only are the functions that can be successfully solved by SamACO presented in the Table. It can be observed that the proposed SamACO performs the best among the four algorithms. SamACO not only has higher successful percentages than the other ant-based algorithms, but also uses a relatively small number of FEs to achieve solutions within the predefined accuracy levels. For example, when solving f_2 (the shifted Schwefel's P2.21), SamACO can find solutions within accuracy levels 1 and 10^{-6} in all runs. However, CACO, COAC, and ACO_R can only obtain solutions within the accuracy level 1 or 10^{-6} successfully in some runs.

2) Comparison With CLPSO and Probability-Learning Algorithms

The scores in Table II show that SamACO generally performs better than CLPSO on the test functions. According to Table III, the average numbers of FEs used by SamACO are

much smaller than those by CLPSO. Furthermore, SamACO can find satisfactory solutions in more test functions than CLPSO.

On the other hand, CMA-ES achieves the smallest *t*-test total scores among the compared algorithms presented in Table II. However, CMA-ES has unsatisfactory performance on functions such as f_{10}, f_{11}, f_{15}, f_{16}. The second place is SamACO, because the other algorithms have positive total scores, meaning that they are not better than SamACO. Table III shows that FEP generally needs more FEs to obtain satisfactory results. CMA-ES can obtain high-quality solutions very fast for most of the test functions.

Fig. 8 illustrates the convergence curves of SamACO and the compared algorithms. It can be seen that SamACO finds solutions with high accuracy very fast for the functions.

3) Analysis on the Computational Complexity of Algorithms

Following the methods in [40], the computational complexity of the algorithms discussed in this paper is computed as in Table IV. Computations of $T0$, $T1$, and $\hat{T}2$ can be referred in [40]. The values of $T1$ and $\hat{T}2$ are obtain on f_7 at n = 30 after 200000 FEs. All of the time values are measured in CPU seconds. The results of SamACO, CACO, COAC, ACO_R, CLPSO, and FEP are obtained on a PC with CPU DualCore 2.0 GHz, RAM 1G, Platform: Visual C++ 6.0. The results of CMA-ES are obtained on CPU Dual 2.0 GHz, RAM 1G, Platform: MATLAB 7.1.0.

By using the Gram-Schmidt process, which is a computationally-heavy operation, the computational complexity of ACO_R is much larger than the other algorithms. In addition, algorithms that use complex distribution models (such as FEP and CMA-ES) generally require a longer computation time. The computational complexity of SamACO is modest, with a complexity value smaller than 3.

VI. Conclusion

An algorithm termed SamACO has been proposed for solving continuous optimization problems in this paper. It demonstrates that after determining candidate variable values, the optimization procedure of ACO in solving discrete optimization problems is in effect a sub-process in solving continuous optimization problems. Therefore, a way to sample promising variable values in the continuous domain and to use pheromones to guide ants' construction behavior has been developed in SamACO to extend ACO to continuous optimization. It is shown that SamACO can also be applied to both discrete optimization problems and mixed discrete-continuous optimization problems. The novelties and characteristics of SamACO are:

1) A new framework for ACO: The SamACO framework extends ACO to continuous optimization, using an effective sampling method to discretize the continuous space, so that the traditional ACO operations for selecting discrete components can be used compatibly.
2) A new sampling method for discretizing the continuous search space: The sampling method aims at balancing memory of previous ants, exploration for new variable values, and exploitation for variable values with high accuracy.
3) Efficient incremental solution construction based on the sampled variable values: Each ant in SamACO constructs solutions incrementally by selecting values from the sampled variable values.

Some state-of-the-art algorithms have been used for comparing the performance with the SamACO algorithm. They include ant-based algorithms for continuous optimization, swarm intelligence algorithms, and representative probability-learning algorithms. The experimental results show that the SamACO algorithm can deliver good performance for both unimodal and multimodal problems.

Appendix I

The SamACO Algorithm

1) /* Initialization */

Initialize parameter values as in Table I.

For i := 1 to n do /* For each variable */

$r_i := (u_i - l_i)/(2m)$ /* Set the initial radius value */

$g_i := m + \vartheta$ /* Set the number of candidate variable value for selection */

For j := 1 to $(m + \vartheta)$ do

Randomly sample variable values according to (2)

$\tau_i^{(j)} := T_0$ /* Assign the initial pheromone value */

End-for

End-for

For j := 1 to $(m + \vartheta)$ do

Evaluate the jth solution according to the objective function

End-for

Sort the m solutions from the best to the worst

Record the best-so-far solution $\{x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)}\}$ and assign each variable i $\tau_i^{(0)} := T_0$

Perform ant's solution construction as the following Step 3)

Perform pheromone update as the following Step 4)

2) /* Generation of the candidate variable values */

/* Perform a dynamic exploitation process to the best-so-far solution */

For i := 1 to n do $g_i := 1$ End-for

flag := False /* Indicate whether exploitation can find a better solution */

For k := 1 to ϑ do /* Repeat for ϑ times */

For i := 1 to n do /* For each variable */

Generate a value $\hat{x}_i$ according to (4)

If ($0 \le q < 1/3$ or $2/3 \le q < 1$)

$x_i^{(m+g_i)} := \hat{x}_i$ /* Record the new variable value */

$\tau_i^{(m+g_i)} := T_0$ /* Assign the initial pheromone value */

$g_i := g_i + 1$ /* Counter adds 1 */

End-if

End-for

Evaluate the solution $\{\hat{x}_1, \hat{x}_2, ..., \hat{x}_n\}$ according to the objective function

If the resulting solution is no worse than the best-so-far solution

For i := 1 to n do $x_i^{(0)} := \hat{x}_i$ End-for /* Update the best-so-far solution */

flag := True

End-if

End-for

If *flag* = True

For i := 1 to n do $r_i := r_i \cdot v_e$ End-for

Else

For i := 1 to n do $r_i := r_i \cdot v_r$ End-for

End-if

/* Perform a random exploration process to the worst solutions*/

For j := $m - \Theta + 1$ to m do

For i := 1 to n do

$x_i^{(j)} := l_i + (u_i - l_i) \cdot rand_i^{(j)}$

$\tau_i^{(j)} := T_0$

End-for

End-for

3) /* Ants' solution construction */

For i := 1 to n do

For k := 1 to m do

Ant k selects the index $l_i^{(k)}$ according to (6) and (7)

$\tilde{x}_i^{(k)} := x_i^{(l_i^{(k)})}$ /* Record the selected variable value */

$\tilde{\tau}_i^{(k)} := \tau_i^{(l_i^{(k)})}$ /* Record the corresponding pheromone value */

End-for

End-for

For k := 1 to m do

For i := 1 to n do

$x_i^{(k)} := \tilde{x}_i^{(k)}$ /* Update the constructed solutions */

$\tau_i^{(k)} := \tilde{\tau}_i^{(k)}$

End-for

Evaluate the kth solution according to the objective function

End-for

4) /* Pheromone update */

Sort the m solutions from the best to the worst

Update the best-so-far solution

Perform pheromone evaporation to the m solutions according to (8)

Perform pheromone reinforcement to the best Ψ solutions according to (9)

For i := 1 to n do $\tau_i^{(0)} := \tau_i^{(1)}$ End-for

5) If (Termination_condition = True)

Output the best solution

Else goto Step 2)

End-if

Appendix II

Benchmark Functions

The test functions used for experiments include the first ten functions for CEC 2005 Special Session on real-parameter optimization [40], as f_5 - f_9 and f_{12} - f_{16} in this paper, and functions from [30] by adding a shifted offset $\boldsymbol{o}$ = (0,1,2,…,n-1), as f_1 - f_4 and f_{10} - f_{11}. The shifted variable vectors and bounds for f_1 - f_4 and f_{10} - f_{11} are defined as $\boldsymbol{x}^{\text{new}} = \boldsymbol{x} - \boldsymbol{o}$, $\boldsymbol{l}^{\text{new}} = \boldsymbol{l} + \boldsymbol{o}$, $\boldsymbol{u}^{\text{new}} = \boldsymbol{u} + \boldsymbol{o}$, where $\boldsymbol{x}$, $\boldsymbol{l}$, $\boldsymbol{u}$ are variable vectors, lower bound, and upper bound defined in [30], $\boldsymbol{x}^{\text{new}}$, $\boldsymbol{l}^{\text{new}}$, $\boldsymbol{u}^{\text{new}}$ are the corresponding shifted vectors. The test functions are as follows.

f_1: Shifted Step

f_2: Shifted Schwefel's P2.21

f_3: Shifted Schwefel's P2.22

f_4: Shifted Quartic with Noise

f_5: Shifted Sphere

f_6: Shifted Schwefel's P1.2

f_7: Shifted Rotated High Conditioned Elliptic

f_8: Shifted Schwefel's P1.2 with Noise

f_9: Schwefel's P2.6 with Global Optimum on Bounds

f_{10}: Shifted Schwefel's P2.26

f_{11}: Shifted Ackley

f_{12}: Shifted Rosenbrock

f_{13}: Shifted Rotated Griewank's without Bounds

f_{14}: Shifted Rotated Ackley's Function with Global Optimum on Bounds

f_{15}: Shifted Rastrigin

f_{16}: Shifted Rotated Rastrigin

Acknowledgment

The authors would like to thank Dr. M. Dorigo, Dr. K. Socha, Dr. P. N. Suganthan, Dr. X. Yao, and Dr. N. Hansen for letting us use their source codes for the benchmark functions. The authors would also like to thank the anonymous associate editor and the referees for their valuable comments to improve the quality of this paper. Jun Zhang is the corresponding author and can be contacted at: junzhang@ieee.org.

References

[1] M. Dorigo, G. D. Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137-172, 1999.

[2] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.

[3] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29-41, Feb. 1996.

[4] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53-66, April 1997.

[5] G. Leguizamón and Z. Michalewicz, "A new version of ant system for subset problems," In *Proc. of IEEE Congress on Evolutionary Computation (CEC'99)*, pp. 1459-1464, Piscataway, 1999.

[6] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," *Trans. Syst. Man Cybern. A., Syst. Humans*, vol. 33, pp. 560-572, Sept. 2003.

[7] S. S. Iyengar, H.-C. Wu, N. Balakrishnan, and S. Y. Chang, "Biologically inspired cooperative routing for wireless mobile sensor networks," *IEEE Syst. J.*, vol. 1, no.1, pp.

29-37, Sept. 2007.

[8] W.-N. Chen and J. Zhang, “An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements,” *IEEE Trans. Syst. Man Cybern. C, Appl. Rev.*, vol. 39, no. 1, pp. 29-43, Jan. 2009.

[9] D. Merkle, M. Middendorf, and H. Schmeck, “Ant colony optimization for resource-constrained project scheduling,” *IEEE Trans. Evol. Comput.*, vol. 6, no.4, pp. 333-346, Aug. 2002.

[10] G. Wang, W. Gong, B. DeRenzi, and R. Kastner, “Ant colony optimizations for resource- and timing-constrained operation scheduling,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 6, pp. 1010-1029, June 2007.

[11] J. Zhang, H. S.-H. Chung, A. W.-L. Lo, and T. Huang, “Extended ant colony optimization algorithm for power electronic circuit design,” *IEEE Trans. Power Electron.*, vol. 24, no. 1, pp. 147-162, Jan. 2009.

[12] N. Monmarché, G. Venturini, and M. Slimane, “On how *pachycondyla apicalis* ants suggest a new search algorithm,” *Future Generation Computer Systems*, vol. 16, pp. 937-946, 2000.

[13] G. Bilchev, and I. C. Parmee, “The ant colony metaphor for searching continuous design spaces,” In *Proc. of AISB Workshop on Evolutionary Computation*, *LNCS* 933, University of Sheffield, UK, Springer-Verlag, Berlin, Germany, pp.25-39, 1995.

[14] M. Wodrich and G. Bilchev, “Cooperative distributed search: the ant’s way,” *Control and Cybernetics*, vol. 3, pp. 413-446, 1997.

[15] M. Mathur, S. B. Karale, S. Priye, V. K. Jayaraman, and B. D. Kulkarni, “Ant colony approach to continuous function optimization,” *Ind. Eng. Chem. Res.*, vol. 39, pp. 3814-3822, 2000.

[16] J. Dréo and P. Siarry, “Continuous interacting ant colony algorithm based on dense

heterarchy," *Future Generation Computer Systems*, vol. 20, pp. 841-856, 2004.

[17] X.-M. Hu, J. Zhang, and Y. Li, "Orthogonal methods based ant colony search for solving continuous optimization problems," *Journal of Computer Science and Technology*, vol. 23, no. 1, pp. 2-18, Jan. 2008.

[18] J. Dréo and P. Siarry, "An ant colony algorithm aimed at dynamic continuous optimization," *Applied Mathematics and Computation*, vol. 181, pp. 457-467, 2006.

[19] H. Huang and Z. Hao, "ACO for continuous optimization based on discrete encoding," M. Dorigo et al. (Eds.), *ANTS 2006, LNCS 4150*, Springer, Berlin, pp. 504-505, 2006.

[20] K. Socha, "ACO for continuous and mixed-variable optimization," M. Dorigo et al. (Eds.), *ANTS 2004, LNCS 3172*, Springer, Berlin, pp. 25-36, 2004.

[21] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, pp. 1155-1173, 2008.

[22] P. Korošec and J. Šilc, "High-dimensional real-parameter optimization using the differential ant-stigmergy algorithm," *International Journal of Intelligent Computing and Cybernetics*, vol. 2, no. 1, pp. 34-51, 2009.

[23] M. Kong and P. Tian, "A direct application of ant colony optimization to function optimization problem in continuous Domain," M. Dorigo et al. (Eds.), *ANTS 2006, LNCS 4150*, Springer, Berlin, pp. 324-331, 2006.

[24] P. Korošec, J. Šilc, K. Oblak, and F. Kosel, "The differential ant-stigmergy algorithm: an experimental evaluation and a real-world application," In *Proc. of IEEE Congress on Evolutionary Computation (CEC'07)*, Singapore, pp. 157-164, 2007.

[25] P. Korošec and J. Šilc, "The differential ant-stigmergy algorithm for large scale real-parameter optimization," M. Dorigo et al. (Eds.), *ANTS 2008, LNCS 5217*, Springer, Berlin, pp. 413-414, 2008.

[26] S. Tsutsui, "An enhanced aggregation pheromone system for real-parameter optimization

in the ACO metaphor," M. Dorigo et al. (Eds.), *ANTS 2006, LNCS 4150,* Springer, Berlin, pp. 60-71, 2006.

[27] F. O. de França, G. P. Coelho, F. J. V. Zuben, and R. R. de F. Attux, "Multivariate ant colony optimization in continuous search spaces," In *Proc. of GECCO'08,* Atlanta, Georgia, USA, pp. 9-16, 2008.

[28] S. H. Pourtakdoust and H. Nobahari, "An extension of ant colony system to continuous optimization problems," M. Dorigo et al. (Eds.), *ANTS 2004, LNCS 3172*, Springer, Berlin, pp. 294-301, 2004.

[29] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 281-295, June 2006.

[30] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82-102, July 1999.

[31] A. Auger and N. Hansen, "Performance evaluation of an advanced local search evolutionary algorithm," *In Proc. of IEEE Congress on Evolutionary Computation (CEC'05),* vol. 2, pp. 1777-1784, 2005.

[32] B. Bullnheimer, R. F. Hartl, and C. Strauss, "A new rank based version of the ant system -- a computational study," *Central European Journal for Operations Research and Economics,* vol. 7, no. 1, pp. 25-38, 1997.

[33] T. Stützle and H. H. Hoos, "MAX-MIN ant system," *Future Generation Computer Systems,* vol. 16, no. 8, pp. 889-914, 2000.

[34] A. C. Zecchin, A. R. Simpson, H. R. Maier, and J. B. Nixon, "Parametric study for an ant algorithm applied to water distribution system optimization," *IEEE Trans. Evol. Comput.,* vol. 9, no. 2, pp. 175-191, April 2005.

[35] M. Guntsch and M. Middendorf, "Pheromone modification strategies for ant algorithms

applied to dynamic TSP," E.J.W Boers et al. (Eds.), *EvoWorkshop 2001*, *LNCS 2037*, pp. 213-222, 2001.

[36] P. Pellegrini, D. Favaretto, and E. Moretti, "On MAX-MIN ant system's parameters," M. Dorigo et al. (Eds.), *ANTS 2006*, *LNCS 4150*, Springer, Berlin, pp. 203-214, 2006.

[37] M. L. Pilat and T. White, "Using genetic algorithms to optimize ACS-TSP," M. Dorigo et al. (Eds.), *ANTS 2002*, *LNCS 2463*, Springer, Berlin, pp. 282-283, 2002.

[38] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*, Morgan Kaufmann, San Mateo, CA, 2001.

[39] M. Clerc and J. Kennedy, "The particle swarm – explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58-73, Feb. 2002.

[40] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization," Nanyang Technol. Univ., Singapore, IIT Kanpur, India, KanGAL Rep. 2005005, May 2005.

Footnote

Manuscript received ... This work was supported in part by the National Natural Science Foundation of China Joint Fund with Guangdong under Key Project U0835002, by the National High-Technology Research and Development Program ("863" Program) of China No. 2009AA01Z208, by the National Science Foundation of China under Project 60975080, by the Sun Yat-Sen Innovative Talents Cultivation Program for Excellent Tutors No. 35000-3126202, and by the Cultivation Fund of the Key Scientific and Technical Innovation Project of the Ministry of Education of China under Grant 706045.

[*]X.-M. Hu and J. Zhang are with the Department of Computer Science, Sun Yat-Sen University, Guangzhou, 510275, China and also with the Key Laboratory of Digital Life (Sun Yat-sen University), Ministry of Education (Jun Zhang is corresponding author, e-mail: junzhang@ieee.org).

[+]H. S.-H. Chung is with the Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong Kong.

[※]Y. Li is with the Department of Electronics and Electrical Engineering, University of Glasgow, Glasgow G12 8LT, Scotland, U.K.

Footnote on Page 15

[1] The MATLAB code used, cmaes.m, Version 2.35, is available at http://www.bionik.tu-berlin.de/user/niko/formersoftwareversions.html

Figure Captions

Table Captions

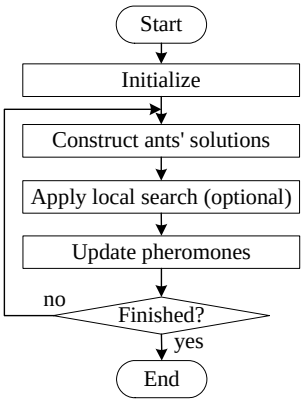

Fig. 1 Framework of the traditional ACO for discrete optimization.

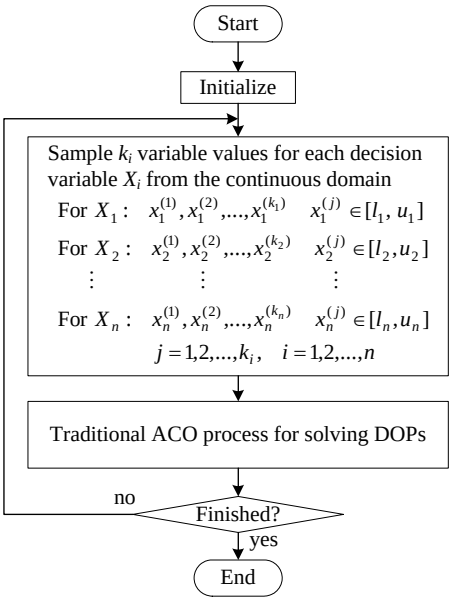

Fig. 2 Framework of the proposed SamACO for continuous optimization.

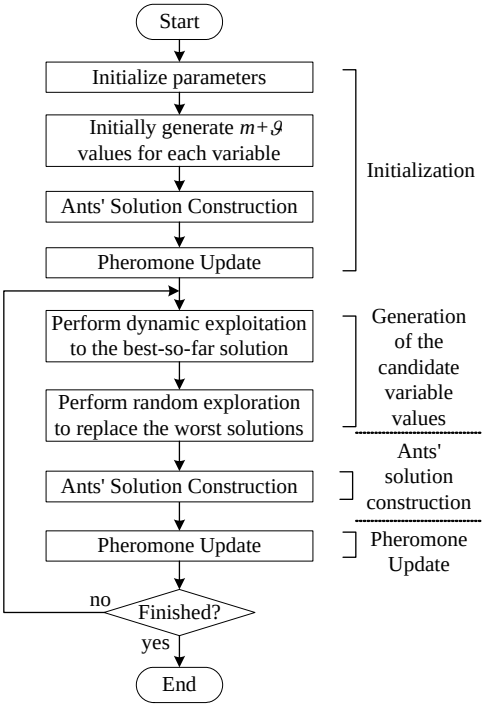

Fig. 3 Flowchart of the SamACO algorithm for continuous optimization.

The best-so-far solution

$$x_1^{(0)} \quad x_2^{(0)} \quad \cdots \quad x_n^{(0)}$$

The best $(m-\Theta)$ solutions constructed by ants

$$\begin{matrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m-\Theta)} & x_2^{(m-\Theta)} & \cdots & x_n^{(m-\Theta)} \end{matrix}$$

For each variable, Θ variable values from the random exploration

$$\begin{matrix} x_1^{(m-\Theta+1)} & x_2^{(m-\Theta+1)} & \cdots & x_n^{(m-\Theta+1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{matrix}$$

For each variable, g_i variable values for each variable X_i from the dynamic exploitation

$$\begin{matrix} x_1^{(m+1)} & x_2^{(m+1)} & \cdots & x_n^{(m+1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m+g_1)} & x_2^{(m+g_2)} & \cdots & x_n^{(m+g_n)} \end{matrix}$$

Fig. 4 Composition of the candidate variable values for the ants to select.

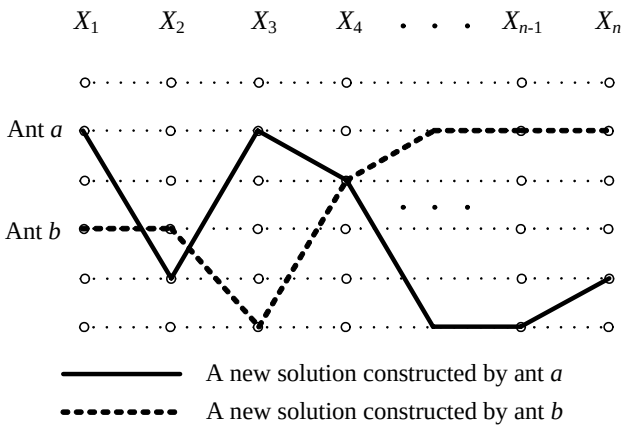

Fig. 5 Illustration of two solutions constructed by ant *a* and ant *b*.

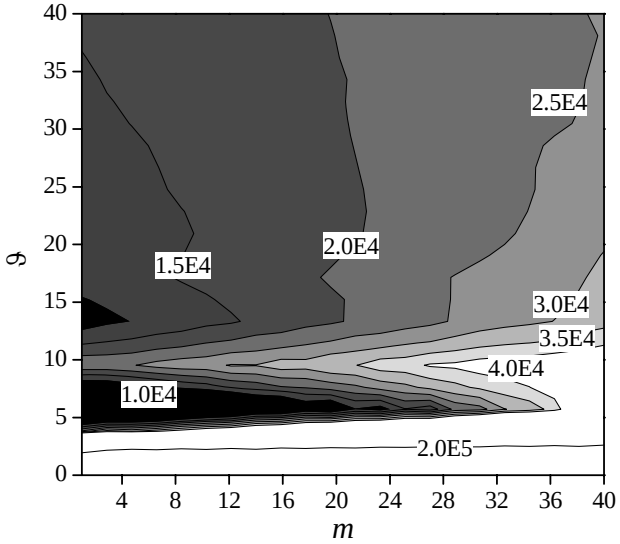

Fig. 6 Contour illustration of using different values of *m* and ϑ on f_6. The contour represents the average number of FEs used for achieving an accuracy level smaller than $\varepsilon = 1$, with the other parameters set as default.

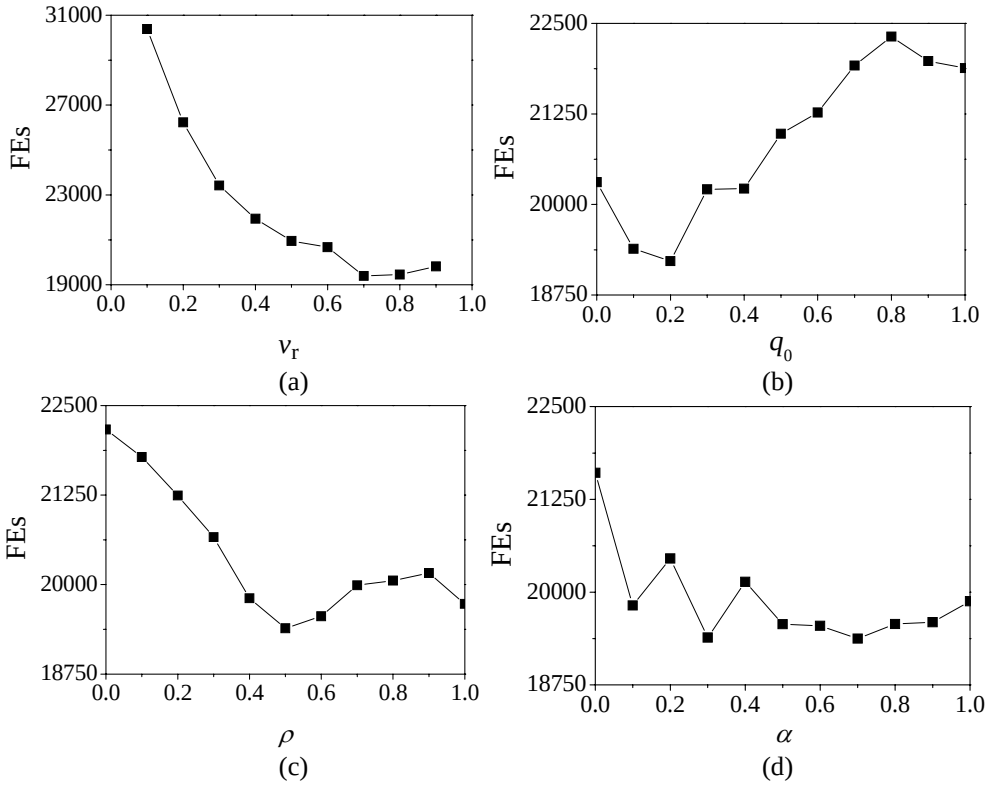

Fig. 7 Illustration of using different values of v_r, q_0, ρ, α on f_6. The y-axis represents the average number of FEs used for achieving an accuracy level smaller than $\varepsilon = 1$, with the other parameters set as default.

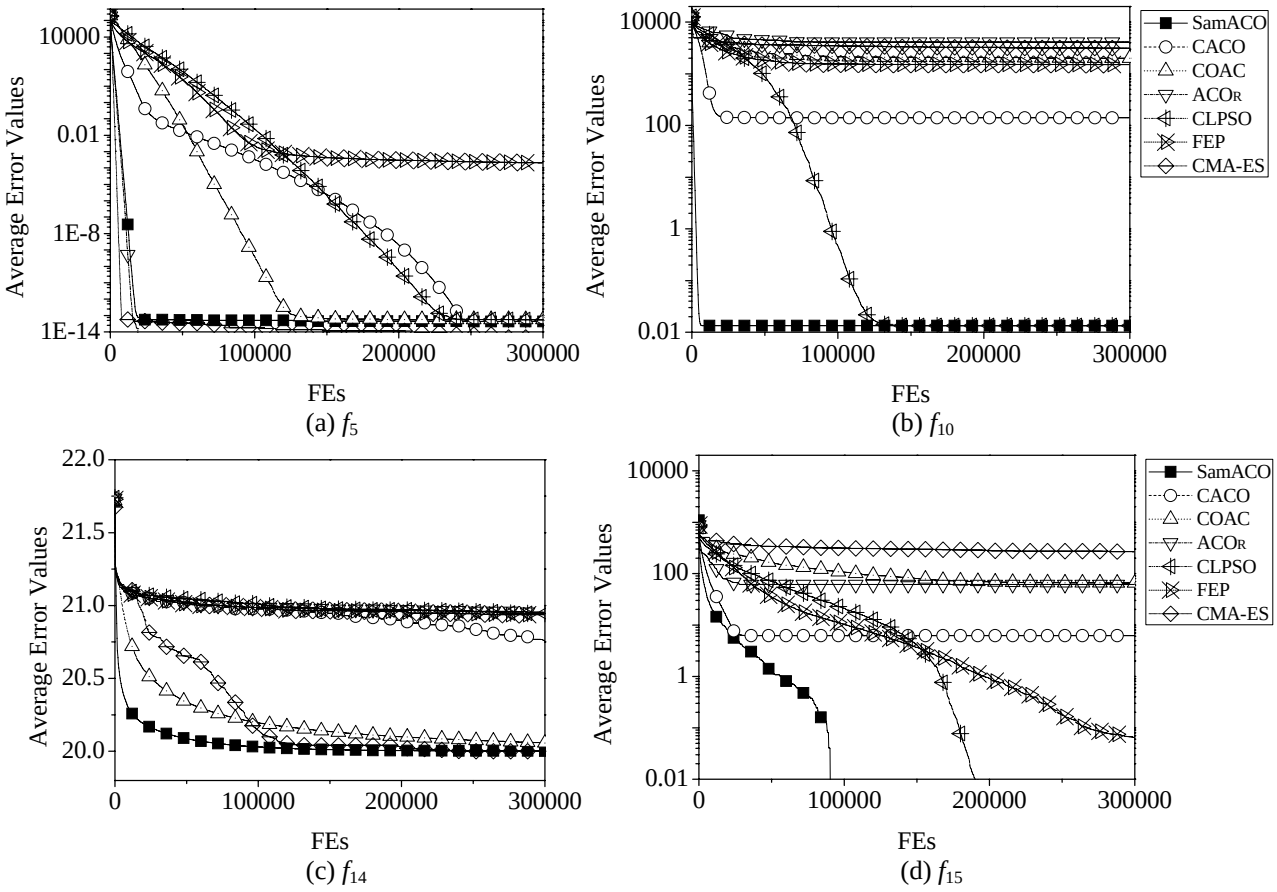

Fig. 8 Convergence graphs of the algorithms. The vertical axis is the average error value, and the horizontal axis is the number of FEs. (a) f_5 (b) f_{10} (c) f_{14} (d) f_{15}

TABLE I
SUMMARY OF PARAMETER SETTINGS IN SAMACO

P	Domain	DS	Summary of Sensitivity Based on DS
m	1.. $+\infty$	20	More ants slow down the convergence, whereas fewer ants reduce the reliability for achieving high-quality solutions
ϑ	1.. $+\infty$	20	Exploitation frequency should be big enough but a too large frequency may induce early convergence
v_r	(0, 1]	0.7	A large radius reduction rate (>0.5) helps better exploit the neighborhood, whereas a small rate (<0.5) may miss promising values
v_e	[1, $+\infty$)	$1/v_r$	A reversal operation of the radius reduction
q_0	[0, 1)	0.1	The probability q_0 is better to be small enough for maintaining diversity and avoiding stagnation
ρ	(0, 1)	0.5	The pheromone evaporation rate is better to be big enough for reducing the attraction of undesirable variable values
α	(0, 1)	0.3	The pheromone reinforcement rate is better to be moderate for increasing the attraction of high-quality variable values and avoiding stagnation
Θ	1..m	1	A small discarding number is enough for introducing diversity to the values
Ψ	1.. ($m - \Theta$)	1	A small elitist number is enough for preserving promising values
T_{max}	(0, $+\infty$)	1.0	The upper and lower limits of pheromone values should be distinguishable with each other
T_{min}	(0,T_{max})	0.1	
T_0	(0,T_{max})	T_{min}	

TABLE II

AVERAGE ERROR VALUES AT n =30, AFTER 300000 FES BY SAMACO, CACO, COAC, ACO_R, CLPSO, FEP, AND CMA-ES

	F		SamACO	CACO [15]	COAC [17]	ACO_R [21]	CLPSO [29]	FEP [30]	CMA-ES[31]
Unimodal	f_1	Mean	0	0†*	0†*	0†*	0†*	0†*	0†*
		St. dev	0	0	0	0	0	0	0
	f_2	Mean	3.55×10^{-15}	1.48†	8.68†	9.97×10^{-7}†	4.80†	1.12×10^{-1}†	1.22×10^{-12}†
		St. dev	0	6.30×10^{-1}	9.09	4.63×10^{-6}	6.63×10^{-1}	1.54×10^{-1}	1.54×10^{-13}
	f_3	Mean	4.74×10^{-3}	8.91×10^{-15}‡	4.84†	3.31†	5.04×10^{-13}‡	5.85×10^{-2}†	1.30×10^{-12}‡
		St. dev	5.80×10^{-3}	1.65×10^{-14}	2.73	2.09	1.31×10^{-13}	4.01×10^{-3}	1.10×10^{-12}
	f_4	Mean	1.30×10^{-2}	1.10×10^{-3}‡	1.18×10^{-1}†	7.93×10^{-3}‡	4.12×10^{-3}‡	7.43×10^{-3}‡	5.99×10^{-2}†
		St. dev	8.57×10^{-3}	5.37×10^{-4}	3.79×10^{-2}	3.98×10^{-3}	9.13×10^{-4}	1.73×10^{-3}	1.83×10^{-2}
	f_5	Mean	4.32×10^{-14}	0‡	5.68×10^{-14}†	0‡	5.46×10^{-14}†	2.06×10^{-4}†	4.55×10^{-15}‡
		St. dev	2.48×10^{-14}	0	0	0	1.14×10^{-14}	2.83×10^{-5}	1.57×10^{-14}
	f_6	Mean	5.68×10^{-14}	676†	3.96×10^{-13}†	0‡	910†	7.99†	4.32×10^{-14}‡
		St. dev	0	115	6.33×10^{-13}	0	208	7.19	2.48×10^{-14}
	f_7	Mean	1.84×10^{5}	7.34×10^{6}†	1.40×10^{6}†	5.95×10^{4}‡	1.53×10^{7}†	3.48×10^{6}†	3.64×10^{-14}‡
		St. dev	4.82×10^{4}	2.11×10^{6}	6.15×10^{5}	3.24×10^{4}	3.22×10^{6}	1.52×10^{6}	2.78×10^{-14}
	f_8	Mean	1.45×10^{4}	5.95×10^{3}‡	8.71×10^{3}‡	1.39×10^{-10}‡	7.23×10^{3}‡	9.17×10^{3}‡	2.94×10^{4}
		St. dev	4.49×10^{3}	1.25×10^{3}	4.06×10^{3}	5.65×10^{-10}	1.06×10^{3}	5.74×10^{3}	4.31×10^{4}
	f_9	Mean	7.04×10^{3}	5.35×10^{3}‡	4.98×10^{3}‡	2.25×10^{3}‡	4.20×10^{3}‡	6.68×10^{3}	1.02×10^{-10}‡
		St. dev	1.66×10^{3}	5.83×10^{2}	1.11×10^{3}	8.20×10^{2}	520	1.66×10^{3}	2.53×10^{-11}
Multimodal	f_{10}	Mean	1.34×10^{-2}	1.41×10^{2}†	2.00×10^{3}†	4.13×10^{3}†	1.34×10^{-2}†	1.51×10^{3}†	3.13×10^{3}†
		St. dev	0	1.35×10^{2}	2.05×10^{2}	7.83×10^{2}	7.93×10^{-13}	410	2.82×10^{2}
	f_{11}	Mean	5.14×10^{-15}	4.72×10^{-13}†	3.36†	3.09†	4.55×10^{-11}†	1.06×10^{-2}†	13.6†
		St. dev	1.92×10^{-15}	3.61×10^{-13}	7.15×10^{-1}	7.39×10^{-1}	1.20×10^{-11}	7.05×10^{-4}	8.68
	f_{12}	Mean	126	1.04×10^{3}†	1.70×10^{3}†	389†	3.81‡	96.5	5.00×10^{-14}‡
		St. dev	294	1.69×10^{3}	3.26×10^{3}	18.2	5.00	236	1.89×10^{-14}
	f_{13}	Mean	1.67×10^{-2}	2.96†	1.52×10^{-2}	2.03×10^{-2}	1.05†	1.21×10^{-1}†	1.36×10^{-14}‡
		St. dev	1.46×10^{-2}	3.00×10^{-1}	1.50×10^{-2}	1.78×10^{-2}	5.00×10^{-2}	1.52×10^{-1}	1.45×10^{-14}
	f_{14}	Mean	20.0	20.8†	20.1†	21.0†	20.9†	20.9†	20.0‡
		St. dev	4.30×10^{-3}	6.13×10^{-2}	2.79×10^{-2}	4.09×10^{-2}	3.48×10^{-2}	5.05×10^{-2}	8.24×10^{-15}
	f_{15}	Mean	1.59×10^{-14}	6.21†	65.2†	61.2†	1.77×10^{-13}†	6.49×10^{-2}†	266†
		St. dev	2.60×10^{-14}	2.06	16.4	14.9	8.09×10^{-14}	1.86×10^{-2}	34.7
	f_{16}	Mean	270	44.0‡	142‡	76.0‡	101‡	99.7‡	587†
		St. dev	86.9	9.44	27.3	22.0	13.0	23.4	204
No. of SamACO significantly better, indifferent, or worse				b† i w‡	b† i w‡	b† i w‡	b† i w‡	b† i w‡	b† i w‡
				10 0 6	12 1 3	8 1 7	10 0 6	11 2 3	7 1 8
Total Scores				4	9	1	4	8	-1

* The *t*-test is made based on the numbers of FEs that are required for achieving an error value 0.

TABLE III

AVERAGE NUMBER OF FES REQUIRED TO ACHIEVE ACCURACY LEVELS SMALLER THAN $\varepsilon_1 = 1$, $\varepsilon_2 = 10^{-6}$ WHEN n =30 FOR SAMACO, CACO, COAC, ACO_R, CLPSO, FEP, AND CMA-ES

	F	ε	SamACO		CACO [15]		COAC [17]		ACO_R [21]		CLPSO[29]		FEP [30]		CMA-ES[31]	
			FEs	%ok	FEs	%ok	FEs	%ok	FEs	%ok	FEs	%ok	FEs	%ok	FEs	%ok
Unimodal	f_1	1	1959	100	10512	100	26861	100	4362	100	69399	100	60104	100	2809	100
		10^{-6}	1959	100	10512	100	26861	100	4362	100	69399	100	60104	100	2809	100
	f_2	1	3425	100	64550	32	50574	12	18224	100	×	0	228000	100	3567	100
		10^{-6}	32388	100	×	0	×	0	157212	96	×	0	×	0	12896	100
	f_3	1	1809	100	8792	100	25059	4	33930	12	61517	100	50860	100	27084	100
		10^{-6}	102278	8	224088	100	×	0	×	0	189904	100	×	0	53550	100
	f_4	1	40	100	2636	100	7574	100	755	100	20945	100	29252	100	878	100
		10^{-6}	×	0	×	0	×	0	×	0	×	0	×	0	×	0
	f_5	1	4481	100	21060	100	39354	100	4055	100	77345	100	64240	100	2393	100
		10^{-6}	10317	100	157464	100	76093	100	9272	100	153155	100	×	0	4742	100
	f_6	1	19394	100	×	0	89542	100	15659	100	×	0	293466	100	8874	100
		10^{-6}	47260	100	×	0	190344	100	37585	100	×	0	×	0	13651	100
Multimodal	f_{10}	1	2782	100	23990	40	×	0	×	0	94518	100	×	0	×	0
		10^{-6}	×	0	×	0	×	0	×	0	×	0	×	0	×	0
	f_{11}	1	21108	100	10684	100	178373	4	×	0	77043	100	65168	100	37613	28
		10^{-6}	42315	100	222992	100	178373	4	×	0	211499	100	×	0	38974	28
	f_{12}	1	158947	4	×	0	×	0	×	0	28462	20	×	0	66292	100
		10^{-6}	×	0	×	0	×	0	×	0	×	0	×	0	73727	100
	f_{13}	1	7395	100	×	0	57781	100	12735	100	289920	20	229324	100	3681	100
		10^{-6}	26806	20	×	0	198101	28	38918	12	×	0	×	0	9012	100
	f_{15}	1	49753	100	×	0	×	0	×	0	165744	100	202920	100	×	0
		10^{-6}	71286	100	×	0	×	0	×	0	230700	100	×	0	×	0

TABLE IV COMPUTATIONAL COMPLEXITY OF ALGORITHMS

Algorithm	$T0$	$T1$	$\hat{T}2$	$(\hat{T}2 - T1)/T0$
SamACO	0.421	2.5	3.434	2.22
CACO [15]	0.421	2.5	4.984	5.90
COAC [17]	0.421	2.5	2.919	0.99
ACO_R [21]	0.421	2.5	845.7	2002.90
CLPSO[29]	0.421	2.5	3.032	1.26
FEP [30]	0.421	2.5	7.897	12.82
CMA-ES[31]	0.406	16.86	21.67	11.85